Online Sparsifying Transform Learning - Part I: Algorithms

Saiprasad Ravishankar†, *Student Member, IEEE,* Bihan Wen†, *Student Member, IEEE,* and Yoram Bresler, *Fellow, IEEE*

***Abstract*—Techniques exploiting the sparsity of signals in a transform domain or dictionary have been popular in signal processing. Adaptive synthesis dictionaries have been shown to be useful in applications such as signal denoising, and medical image reconstruction. More recently, the learning of sparsifying transforms for data has received interest. The sparsifying transform model allows for cheap and exact computations. In this work, we develop a methodology for online learning of square sparsifying transforms. Such online learning can be particularly useful when dealing with big data, and for signal processing applications such as real-time sparse representation and denoising. The proposed transform learning algorithms are shown to have a much lower computational cost than online synthesis dictionary learning. In practice, the sequential learning of a sparsifying transform typically converges faster than batch mode transform learning. Preliminary experiments show the usefulness of the proposed schemes for sparse representation, and denoising.**

***Index Terms*—Sparse representations, Sparsifying transforms, Online learning, Big data, Dictionary learning, Image representation, Denoising, Machine learning.**

I. Introduction

The sparsity of natural signals in a certain transform domain or dictionary has been widely exploited in various applications in recent years. Various sparse models have been studied such as the *synthesis model* [1], *analysis model* [1], [2], and *transform model* [3], [4]. The adaptation of these models to a class of signals, known as data-driven learning, or learning in short, has also been studied [4]–[8], and shown to be useful in applications. In particular, the learning of the transform model has been shown to be highly efficient compared to the other models [4], [9]. In this work, we develop a methodology for online learning of square sparsifying transforms. Such online learning can be particularly useful when dealing with big data, and for signal processing applications such as real-time sparse representation and denoising. In the following, we briefly discuss the various sparse signal models and their learning, and then present the main contributions of this work.

A. Sparse Models and their Learning

The popular *synthesis model* suggests that a signal $y \in \mathbb{R}^n$ can be sparsely represented in a dictionary $D \in \mathbb{R}^{n \times K}$ as $y = Dx$, where $x \in \mathbb{R}^K$ is sparse, i.e., $\|x\|_0 \ll K$. The ℓ_0 "norm" counts the number of non-zeros in x. Natural signals usually satisfy $y = Dx + e$, where e is an approximation error in the signal domain [10]. Given a signal y and dictionary D, the synthesis sparse coding problem finds x that minimizes $\|y - Dx\|_2^2$ subject to $\|x\|_0 \leq s$, where s is the required synthesis sparsity level. This sparse coding problem is in general, however, NP-hard (Non-deterministic Polynomial-time hard) [11], [12]. Various algorithms [13]–[20] have been proposed to solve this problem. While some of these algorithms are guaranteed [15], [18], [21]–[24] to provide the correct solution under certain conditions, these conditions are often violated in applications. Moreover, these algorithms are typically computationally expensive for large-scale problems.

The alternative *analysis model* [1] for a signal y says that Ωy is sparse ($\|\Omega y\|_0 \ll m$), where $\Omega \in \mathbb{R}^{m \times n}$ is an analysis dictionary [25]. A more general *noisy signal analysis model* [25], [26] models the signal y as $y = z + e$, with Ωz sparse, and e a (small) noise term in the signal domain. Given the analysis dictionary Ω, the problem of recovering the clean signal z from the noisy y involves minimizing $\|y - z\|_2^2$ subject to $\|\Omega z\|_0 \leq m - l$, where l is referred to as the co-sparsity level (number of zeros) [25]. This problem is known as the analysis sparse coding problem [25], and is also NP-hard, just like synthesis sparse coding. Approximate algorithms have been proposed for solving the analysis sparse coding problem [2], [8], [25], [27], [28], which similar to the synthesis case are computationally expensive.

In this work, we focus our attention on the classical transform model, which suggests that a signal y is *approximately sparsifiable* using a transform $W \in \mathbb{R}^{m \times n}$, that is $Wy = x + e$ where $x \in \mathbb{R}^m$ is sparse in some sense, and e is a small residual in the transform rather than in the signal domain. When $m = n$, the transform $W \in \mathbb{R}^{n \times n}$ is called a square transform. Otherwise, it is considered overcomplete. Natural signals are known to be approximately sparse in various transform domains such as the discrete cosine transform (DCT), Wavelets [29], and Curvelets [30].

The transform model is more general than both the analysis and noisy signal analysis models (cf. [1], [4] for the distinctions between the various models). Importantly, it allows for much faster computations than the synthesis and noisy signal analysis models. Given a signal y and sparsifying transform W, the process of obtaining a sparse code x of sparsity s involves minimizing $\|Wy - x\|_2^2$ subject to $\|x\|_0 \leq s$. The solution $\hat{x}$ here is obtained exactly and cheaply by zeroing

† Equal contributors. This work was supported in part by the National Science Foundation (NSF) under grants CCF-1018660 and CCF-1320953.

S. Ravishankar, B. Wen, and Y. Bresler are with the Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois, Urbana-Champaign, IL, 61801 USA e-mail: (ravisha3, bwen3, ybresler)@illinois.edu.

out all but the s coefficients of largest magnitude in Wy. In contrast, sparse coding with synthesis or analysis dictionaries involves solving NP-hard problems approximately. Given the transform W and sparse code x, we can also obtain a (simple) least squares estimate of the signal y by minimizing $\|Wy - x\|_2^2$ over $y \in \mathbb{R}^n$. The recovered signal is $\hat{y} = W^\dagger x$, where $W^\dagger$ is the pseudo-inverse of W.

Recently, the adaptation of sparse models to data has received much attention [4]–[8], [25], [31]–[39]. Various applications such as denoising, and compressed sensing benefit from an adaptive sparse model. Importantly, the learning of transform models has been shown to be much cheaper than synthesis, or analysis dictionary learning [4], [40]. Adaptive transforms also provide competitive, or useful signal reconstruction quality in applications [4], [40]–[44].

B. Online Learning and Big Data

Prior work on transform learning focused on batch learning [4], [9], where the sparsifying transform is adapted using all the training data simultaneously. However, for big data, the dimensions of the training data set are typically very large. Hence, batch learning of a sparsifying transform using existing alternating algorithms [9] is computationally expensive in both time and memory, and may be even infeasible. Moreover, in real-time applications, the data arrives sequentially, and must also be processed sequentially to limit latency. Thus, this setting renders batch learning infeasible, since in real-time applications, one does not have access to all the data at once. To address these problems, we introduce in this work a scheme for online, or sequential learning of a square sparsifying transform.

Our framework adapts sequentially the sparsifying transform and sparse codes (and/or signal estimates) for signals (or, measurements) that arrive, or are processed sequentially. Such online/sequential transform learning is amenable to big data, and applications such as real-time sparse representation (compression), denoising, and compressed sensing. As we show in this work, online transform learning involves cheap computations and modest memory requirements. Moreover, in Part II of this work [45], convergence guarantees are provided for online transform learning. Our numerical experiments illustrate the usefulness of our schemes for big data processing (online sparse representation and denoising). As we show, the sequential transform learning scheme can also converge faster than the batch transform learning scheme [4], [9], [46] in practice.

While the online learning of synthesis dictionaries has been studied previously [36], [47]–[50], the online adaptation of the transform model allows for much cheaper computations. Furthermore, the proof by Mairal et al. [47] of the convergence of online synthesis dictionary learning requires various restrictive assumptions. In contrast, the analysis in Part II (see [45] for details) relies on only a few simple assumptions. Another feature distinguishing our formulation is that in the previous work [47], the objective is biconvex, so that the non-convexity in the problem vanishes when a particular variable is kept fixed. This is not the case in our formulation, in which the non-convexity is due to the ℓ_0 "norm" and the log determinant terms. Our formulation remains non-convex even when one of the variables is fixed.

Other very recent works consider synthesis dictionary learning for big data. Wang et al. [51] propose a scheme to incrementally add new columns to the learned dictionary for every new block of signals (sequentially) processed. However, the dictionary size in this method grows continuously (as more blocks of signals are processed), which is undesirable. Another recent work on synthesis dictionary learning for big data is a split and merge learning algorithm [52], which however, is not an online algorithm. The big dataset is split into subsets, and dictionaries are learnt in parallel for each subset, before being merged to a single smaller dictionary. However, as the size of the big dataset increases, either the size of each subset increases monotonically, or the final merging step becomes more complex, requiring increasing time and memory. Although faster than conventional dictionary learning, the dictionary learned by this method [52] is worse.

We organize the rest of this paper as follows. Section II describes the prior work on batch transform learning, and then presents our proposed problem formulations for online and mini-batch (that handles blocks of signals sequentially) transform learning and denoising. In Section III, we present efficient algorithms to solve our proposed problem formulations, and discuss our algorithms' computational, latency, and memory advantages. Section IV provides experimental results demonstrating the convergence and computational properties of the proposed schemes. We also show results for sparse representation and denoising. In Section V, we conclude.

II. Transform Learning Problem Formulations

A. Batch Learning

In batch learning, the sparsifying transform is adapted to all the training data simultaneously. Given a matrix $Y \in \mathbb{R}^{n \times N}$, whose columns y_i $(1 \leq i \leq N)$ represent all the training signals, the problem of learning an adaptive square sparsifying transform W (in batch mode) is formulated as follows [4], [9]

$$\text{(P0)} \quad \min_{W,X} \|WY - X\|_F^2 + \lambda v(W) \quad s.t. \quad \|x_i\|_0 \leq s \ \forall \ i$$

where x_i denotes the i^{th} column of the sparse code matrix X, s is a given sparsity level, and $v(W) = -\log |\det W| + \|W\|_F^2$. The term $\|WY - X\|_F^2$ in (P0) is the *sparsification error* for the data Y in the transform W. The sparsification error is the modeling error in the transform model, and hence we minimize it in order to learn the best possible transform model.

Problem (P0) also has $v(W)$ as a regularizer in the objective to prevent trivial solutions [4]. Specifically, the log determinant penalty enforces full rank on the transform W, and eliminates degenerate solutions such as those with zero, or repeated rows. The $\|W\|_F^2$ penalty helps remove a 'scale ambiguity' [4] in the solution (the scale ambiguity occurs when the data admits an exactly sparse representation). Together, the log determinant and Frobenius norm penalty terms fully control the condition number of the learnt transform [4]. This eliminates badly conditioned transforms, which typically convey little information and may degrade performance in applications. As

shown in [4], the condition number of the transform $\kappa(W)$ can be upper bounded by a monotonically increasing function of $v(W)$. Hence, minimizing $v(W)$ encourages reduction of the condition number. In the limit $\lambda \to \infty$, the condition number of the optimal transform(s) in (P0) tends to 1. In practice, the transforms learnt via (P0) have condition numbers close to 1 even for finite λ [9], [46]. The specific choice of λ depends on the application and desired condition number. The regularizer $v(W)$ also penalizes bad scalings. Given a transform W and a scalar $\alpha \in \mathbb{R}$, $v(\alpha W) \to \infty$ as the scaling $\alpha \to 0$ or $\alpha \to \infty$.

To make the two terms in the cost of (P0) scale similarly, we set $\lambda = \lambda_0 \|Y\|_F^2$ with constant $\lambda_0 > 0$. We have shown that the cost function in (P0) is lower bounded [4] by $\frac{n\lambda}{2} + \frac{n\lambda}{2}\log(2) > 0$. The minimum objective value for Problem (P0) equals this lower bound if and only if there exists a pair $(\hat{W}, \hat{X})$ with $\hat{X}$ whose columns have sparsity $\leq s$ and $\hat{W}$ whose (non-zero) singular values are all equal, such that $\hat{W}Y = \hat{X}$.

Problem (P0) admits an equivalence class of solutions/minimizers. Given a particular minimizer $(\tilde{W}, \tilde{X})$, we can form equivalent minimizers by simultaneously permuting the rows of $\tilde{W}$ and $\tilde{X}$, or by pre-multiplying them by a diagonal ± 1 sign matrix [4]. More generally, because the objective in (P0) is unitarily invariant, then given a minimizer $(\tilde{W}, \tilde{X})$, the pair $(\Xi\tilde{W}, \Xi\tilde{X})$ is another equivalent minimizer for all sparsity-preserving *orthonormal* matrices Ξ, i.e., orthonormal Ξ satisfying $\|\Xi\tilde{x}_i\|_0 \leq s \ \forall \ i$.

B. Online Learning

We now introduce our problem formulation for online sparsifying transform learning. The goal here is to adapt the transform and sparse code to data that arrive, or are processed sequentially. For time $t = 1, 2, 3, ...$, the optimization problem to update the sparsifying transform and sparse code based on new data $y_t \in \mathbb{R}^n$ is as follows:

$$(\text{P1}) \ \left\{\hat{W}_t, \hat{x}_t\right\} = \underset{W,\, x_t}{\arg\min} \frac{1}{t}\sum_{j=1}^{t}\left\{\|Wy_j - x_j\|_2^2 + \lambda_j v(W)\right\}$$
$$s.t. \ \ \|x_t\|_0 \leq s, \ x_j = \hat{x}_j, \ 1 \leq j \leq t-1$$

where $\lambda_j = \lambda_0 \|y_j\|_2^2 \ \forall \, j$, $\hat{W}_t$ is the optimal transform at time t, and $\hat{x}_t$ is the optimal sparse code for y_t. Note that only the latest sparse code is updated at time t. The condition $x_j = \hat{x}_j, \ 1 \leq j \leq t-1$, is therefore assumed. For brevity, we will not explicitly restate this condition (or, its appropriate variant) in the formulations in the rest of this paper. On the other hand, at each time t the transform $\hat{W}_t$ is optimized using all the data $\{y_j\}_{j=1}^t$ and sparse codes $\{x_j\}_{j=1}^t$ up to time t. Problem (P1) is simply an online version of the batch problem (P0), and hence it shares some properties with (P0). Specifically, the constant λ_0 controls the condition number of the learned transform.

Although Problem (P1) outputs an optimal $\hat{W}_t$ for each t, it is typically impractical to store (in memory) $\hat{W}_t$ for all t. In our experiments, we store only the latest $\hat{W}_t$, and use it as an initialization for the algorithm that solves for $\hat{W}_{t+1}$. At any time instant t, one can obtain a least squares estimate of the signals $\{y_j\}_{j=1}^t$ from their sparse codes as $\left\{\hat{W}_t^{-1}\hat{x}_j\right\}_{j=1}^t$ (i.e., 'decompressing' the signals from stored sparse codes).

For small values of t, Problem (P1) may highly overfit the transform to the data. This is typically undesirable. In order to overcome this problem, for small values of t, we only perform an update of the sparse codes (with a fixed W – set to a reasonable initialization).

Problem (P1) can be further modified, or improved in certain scenarios. For example, for non-stationary data, it may not be possible to fit a single transform W to y_t for all t. In this case, one can introduce a forgetting factor ρ^{t-j} (with a constant $0 < \rho < 1$), that scales the terms in (P1). Such a forgetting factor would diminish the influence of "old" data. The objective function (within the minimization) in (P1) is then modified as

$$\frac{1}{t}\sum_{j=1}^{t}\rho^{t-j}\left\{\|Wy_j - x_j\|_2^2 + \lambda_j v(W)\right\} \tag{1}$$

Note that this is only one form of the forgetting factor (cf. [47] for another form).

For fixed size data sets, Problem (P1) can be used as an effective sequential learning and sparse coding (compression) strategy. In this case, it is typically useful to make multiple passes through the data set to overcome the causality restriction on the update of the sparse codes. In this case, the same training signals are used, or examined multiple times by (P1), which crucially allows to better update the sparse code using a transform determined by the entire data. Similar strategies have been proposed for online synthesis dictionary learning [47].

C. Mini-batch Learning

A useful variation of online learning is mini-batch learning [47], where we process more than one signal at a time. Mini-batch learning may provide potential reduction in operation count over online learning. However, the processing of blocks of signals leads to increased latency, and memory requirements.

Assuming a fixed block size (or, mini-batch size) of M, the J^{th} ($J \geq 1$) block of signals (in terms of the time sequence $\{y_t\}$) is $Y_J = \left[y_{JM-M+1} \mid y_{JM-M+2} \mid \ \ ... \ \ \mid y_{JM}\right]$. For $J = 1, 2, 3, ...$, the mini-batch sparsifying transform learning problem is formulated as follows:

$$\left\{\hat{W}_J, \hat{X}_J\right\} = \underset{W,\, X_J}{\arg\min} \frac{1}{JM}\sum_{j=1}^{J}\left\{\|WY_j - X_j\|_F^2 + \Lambda_j v(W)\right\}$$
$$s.t. \ \ \|x_{JM-M+i}\|_0 \leq s \ \ \forall \, i \in \{1, .., M\} \ \ (\text{P2})$$

where the weight $\Lambda_j = \lambda_0 \|Y_j\|_F^2$, and the matrix $X_J = \left[x_{JM-M+1} \mid x_{JM-M+2} \mid \ \ ... \ \ \mid x_{JM}\right]$ contains the block of sparse codes corresponding to the block Y_J.

Note that both Problems (P1) and (P2) handle signals sequentially, or involve sequential learning. However, (P1) handles one signal at a time, whereas (P2) uses blocks of signals at a time. In order to clearly distinguish between these two cases in the rest of this paper, we will use the terminology

'online learning' to refer to only the case where one signal is processed at a time instant, and we use 'mini-batch learning' to explicitly refer to the case $M > 1$.

D. Online Denoising Formulation

Online (and mini-batch) transform learning could be used for various applications such as sparse representation (compression), denoising, compressed sensing, etc. Here, we consider an extension of (P1) and (P2) (which by themselves, can be used for sparse representation of signals) to denoising. Denoising aims to recover an estimate of the signal $z \in \mathbb{R}^n$ from its measurement $y = z + h$, corrupted by noise h. Here, we consider a time sequence of measurements $\{y_t\}$, with $y_t = z_t + h_t$, and $h_t \in \mathbb{R}^n$ being the noise. We assume h_t whose entries are independent identically distributed (i.i.d.) Gaussian with zero mean and variance σ^2. The goal of online denoising is to recover estimates of $z_t \ \forall \ t$. We model the underlying noiseless signals z_t as approximately sparse in a (unknown) transform domain.

Previous work [4], [40] presented a formulation for adaptive sparsifying transform-based batch denoising. Here, we instead present a simple denoising formulation that is a modification of the online learning Problem (P1). For $t = 1, 2, 3, ...$, we solve

$$\text{(P3)} \ \min_{W,\, x_t} \frac{1}{t} \sum_{j=1}^{t} \left\{ \|Wy_j - x_j\|_2^2 + \lambda_j v(W) + \tau_j^2 \|x_j\|_0 \right\}$$

where the weights $\tau_j \propto \sigma$. Problem (P3) is to estimate $\hat{W}_t$, and $\hat{x}_t$, and the denoised signal is computed simply as $\hat{z}_t = \hat{W}_t^{-1} \hat{x}_t$. Similar to the extension of (P1) to (P2), we can also extend (P3) to its mini-batch version. The different variations of (P1) suggested in Section II-B (such as forgetting factor, and multiple passes) can also be applied here.

Problem (P3) can also be used for patch-based denoising of large images [6], [40], or image sequences. The overlapping patches of the noisy images are processed sequentially, and the denoised image is obtained by averaging the denoised patches at their respective image locations.

III. Algorithms and Properties

A. Batch Transform Learning

Previous work [4], [9], [46] proposed an alternating algorithm for solving Problem (P0) that alternates between solving for X (*sparse coding step*) and W (*transform update step*), with the other variable kept fixed. The sparse coding step is as follows

$$\min_{X} \|WY - X\|_F^2 \ \ s.t. \ \ \|x_i\|_0 \leq s \ \ \forall \ i \tag{2}$$

The above problem is to project each column of WY onto the (non-convex) set of vectors with sparsity $\leq s$. An optimal solution [4], [40] to (2) is computed exactly as $\hat{x}_i = H_s(Wy_i) \ \forall \, i$, where the operator $H_s(\cdot)$ zeros out all but the s coefficients of largest magnitude in a vector. If there is more than one choice for the s coefficients of largest magnitude in a vector z, which can occur when multiple entries in z have identical magnitude, then we choose $H_s(z)$ as the projection of z for which the indices of the s largest magnitude elements in z are the lowest possible.

The transform update step of (P0) involves the following unconstrained non-convex minimization.

$$\min_{W} \ \|WY - X\|_F^2 + \lambda v(W) \tag{3}$$

The solution to (3) is also computed in closed-form [9], [46]. Let us factorize $YY^T + \lambda I$ (where I is the identity matrix) as LL^T (e.g., the Cholesky factor, or the positive definite square root [1] L), with $L \in \mathbb{R}^{n \times n}$. Further, let $L^{-1} Y X^T = Q \Sigma R^T$ be a full singular value decomposition (SVD), where Q, Σ, and R are $n \times n$ matrices. Then, a global minimizer of (3) is given as

$$\hat{W} = 0.5R \left(\Sigma + \left(\Sigma^2 + 2\lambda I \right)^{\frac{1}{2}} \right) Q^T L^{-1} \tag{4}$$

where the $(\cdot)^{\frac{1}{2}}$ in (4) denotes the positive definite square root. The solution above is unique if and only if YX^T is non-singular [46]. Furthermore, the solution is invariant to the choice of factor L.

We now discuss the computational cost, and memory requirements of this batch transform learning algorithm. The total cost per iteration (of sparse coding and transform update) of the batch transform learning algorithm scales (assuming $n \ll N$) as $O(Nn^2)$. This is much lower than the per-iteration cost of learning an $n \times K$ overcomplete ($K > n$) synthesis dictionary D using K-SVD [5], which scales (assuming that the synthesis sparsity level $s \propto n$, and $K \propto n$) as $O(Nn^3)$. Previous work [4], [9], [46] has demonstrated that batch transform learning also converges quickly (in a small number of iterations) in practice [2]. The memory requirement of batch transform, or dictionary learning scales as $O(Nn)$. This cost becomes prohibitive for large N.

B. Online Transform Learning

Here, we solve Problem (P1) at each time instant t by alternating minimization (similar to (P0)).

1) Sparse Coding: In the sparse coding step, we solve (P1) for x_t with fixed $W = \hat{W}_{t-1}$ (warm start) as follows:

$$\min_{x_t} \ \|Wy_t - x_t\|_2^2 \ \ s.t. \ \ \|x_t\|_0 \leq s \tag{5}$$

The sparse coding solution is given as $\hat{x}_t = H_s(Wy_t)$, with $H_s(\cdot)$ defined as in Section III-A.

2) Exact Transform Update: In the transform update step, we solve (P1) with fixed $x_j = \hat{x}_j, \ 1 \leq j \leq t$, as follows:

$$\min_{W} \frac{1}{t} \sum_{j=1}^{t} \left\{ \|Wy_j - x_j\|_2^2 + \lambda_j v(W) \right\} \tag{6}$$

This problem has a closed-form solution (similar to (4)). Let $t^{-1} \sum_{j=1}^{t} \left(y_j y_j^T + \lambda_j I \right) = L_t L_t^T$ (e.g., the positive definite or

[1] This is nothing but the well-known eigenvalue decomposition square root.

[2] Moreover, it is guaranteed to converge to (at least) a local minimum of the objective [46].

eigenvalue decomposition (EVD) square root L_t). We compute the full SVD of $L_t^{-1}\Theta_t = Q_t\Sigma_t R_t^T$, where

$$\Theta_t = t^{-1}\sum_{j=1}^{t} y_j \hat{x}_j^T \tag{7}$$

Then, a closed-form solution [3] to (6) is given as

$$\hat{W}_t = 0.5R_t\left(\Sigma_t + \left(\Sigma_t^2 + 2\beta_t I\right)^{\frac{1}{2}}\right)Q_t^T L_t^{-1} \tag{8}$$

We can compute $\Gamma_t \triangleq t^{-1}\sum_{j=1}^{t} y_j y_j^T$, Θ_t, and $\beta_t \triangleq \sum_{j=1}^{t} t^{-1}\lambda_j$ sequentially over time. However, computing the inverse square root L_t^{-1}, the matrix-matrix product $L_t^{-1}\Theta_t$, and its full SVD would all cost $O(n^3)$ computations. Instead, we propose a computationally cheaper transform update algorithm as follows.

3) Efficient Approximate Transform Update: The following algorithm involves efficient SVD computations, and eliminates matrix-matrix multiplications. To compute the transform update solution, we first efficiently factorize $t^{-1}\sum_{j=1}^{t}\left(y_j y_j^T + \lambda_j I\right)$ as $L_t L_t^T$ (i.e., take square root), with $L_t \in \mathbb{R}^{n\times n}$. Here, we work with the eigenvalue decomposition (EVD) square root. Denoting the full EVD of $\Gamma_{t-1} = (t-1)^{-1}\sum_{j=1}^{t-1} y_j y_j^T$ as $U_{t-1}\Delta_{t-1}U_{t-1}^T$, or alternatively as the triplet $(U_{t-1}, \Delta_{t-1}, U_{t-1})$, the full EVD of $\Gamma_t = (1-t^{-1})\Gamma_{t-1} + t^{-1}y_t y_t^T$ can be found via a rank-1 update to the (scaled) EVD of Γ_{t-1} [53], [54]. We denote this as $(U_t, \Delta_t, U_t) = \mathcal{U}\left[U_{t-1}, (1-t^{-1})\Delta_{t-1}, U_{t-1}, A_t\right]$, where $A_t = t^{-1}y_t y_t^T$ is a rank-1 matrix. Here, $\mathcal{U}(U,\Sigma,V,A)$ denotes the rank-1 SVD (in the aforementioned case, this is just the EVD) update operation that produces the full SVD (U',Σ',V') of matrix $U\Sigma V^T + A$, where A is a rank-1 term and $U\Sigma V^T$ is a known SVD [53], [54]. Next, let $\beta_{t-1} = \sum_{j=1}^{t-1}(t-1)^{-1}\lambda_j$. Then, $\beta_t = (1-t^{-1})\beta_{t-1} + t^{-1}\lambda_t$. The EVD square root of $t^{-1}\sum_{j=1}^{t}\left(y_j y_j^T + \lambda_j I\right)$ is then computed as $L_t = U_t\left(\Delta_t + \beta_t I\right)^{\frac{1}{2}}U_t^T$ and its inverse is $L_t^{-1} = U_t\left(\Delta_t + \beta_t I\right)^{-\frac{1}{2}}U_t^T$.

The matrix-matrix products in the formula for L_t^{-1} are not explicitly computed. Instead, we will only need the application of L_t^{-1} to a vector, which can be performed efficiently with $O(n^2)$ computation by applying U_t^T, the diagonal matrix $\left(\Delta_t + \beta_t I\right)^{-\frac{1}{2}}$, and U_t in succession.

In order to compute the closed-form solution to (6), we need to also compute the full SVD of $t^{-1}\sum_{j=1}^{t} L_t^{-1}y_j\hat{x}_j^T = L_t^{-1}\Theta_t$. In order to simplify this computation, we perform the following approximation:

$$\begin{aligned} L_t^{-1}\Theta_t &= L_t^{-1}\left\{(1-t^{-1})\Theta_{t-1} + t^{-1}y_t\hat{x}_t^T\right\} &(9)\\ &\approx (1-t^{-1})L_{t-1}^{-1}\Theta_{t-1} + t^{-1}L_t^{-1}y_t\hat{x}_t^T &(10)\end{aligned}$$

With the above approximation, and the fact that $t^{-1}L_t^{-1}y_t\hat{x}_t^T$ is a rank-1 matrix, the estimate of the full SVD of $L_t^{-1}\Theta_t$ can be obtained by performing a rank-1 update (similar to that for Γ_t) to the scaled (by $1-t^{-1}$) SVD estimate of $L_{t-1}^{-1}\Theta_{t-1}$. Denoting the SVD estimate of $L_t^{-1}\Theta_t$ as $Q_t\Sigma_t R_t^T$, we have that $(Q_t,\Sigma_t,R_t) = \mathcal{U}\left[Q_{t-1}, (1-t^{-1})\Sigma_{t-1}, R_{t-1}, \tilde{A}_t\right]$, where $\tilde{A}_t = t^{-1}L_t^{-1}y_t\hat{x}_t^T$ is a rank-1 matrix.

Now, once the full SVD estimate of $L_t^{-1}\Theta_t$ is computed as $Q_t\Sigma_t R_t^T$ (compute only the matrices in this decomposition, not the products), the (estimate of) closed-form solution for Problem (6) is simply

$$\hat{W}_t = 0.5R_t\left(\Sigma_t + \left(\Sigma_t^2 + 2\beta_t I\right)^{\frac{1}{2}}\right)Q_t^T L_t^{-1} \tag{11}$$

Again, we do not perform any of the matrix-matrix multiplications in (11). Instead, we store the individual matrices, and apply them one by one on vectors, at a computational cost of $O(n^2)$.

Note that the only approximation in the above algorithm arises in (10). Otherwise all rank-1 updates above can be performed up to machine precision accuracy. The net error in the approximation in (10) at time t (i.e., the difference between $L_t^{-1}\Theta_t$ and its SVD estimate at time t [4]) is given as $E_t = \sum_{j=2}^{t}\frac{j-1}{t}\Upsilon_j$, where $\Upsilon_j = \left(L_j^{-1} - L_{j-1}^{-1}\right)\Theta_{j-1}$. The proof of this result is in Appendix A. In the formula of E_t, the Υ_j for smaller j values gets scaled by smaller numbers (i.e., $(j-1)/t$). It follows from results in Part II [45] (based on the bound for the rightmost term in equation (24) of [45]) that Υ_j decays (in norm) as C/j, for some constant C. Based on that result, it is easy to show that the approximation error E_t is bounded by C for all t.

In order to prevent undesirable error accumulations over time, one may monitor the relative error $\|E_t\|_F \,/\, \left\|L_t^{-1}\Theta_t\right\|_F$, and reset the computation as shown below. The relative error can be shown to be upper bounded (up to a scale factor [5]) by $\sum_{j=2}^{t}(\left\|L_j^{-1} - L_{j-1}^{-1}\right\|_F \,/\, \left\|L_t^{-1}\right\|_F)$. Since we only store the eigenvectors and eigenvalues of L_j^{-1} rather than L_j^{-1} itself, we further easily bound the term $\left\|L_j^{-1} - L_{j-1}^{-1}\right\|_F$ in the preceding expression by $\|U_jS_j - U_{j-1}S_{j-1}\|_F + \min\left\{\|(U_j - U_{j-1})S_j\|_F, \|(U_j - U_{j-1})S_{j-1}\|_F\right\}$, where the matrix U_j was defined previously [6] and $S_j \triangleq \left(\Delta_j + \beta_j I\right)^{-1/2}$. We thus have a simple upper bound for the relative error that is cheap to compute (can be computed sequentially over time) at $O(n^2)$ cost, and can be monitored. If it rises above a threshold ϵ, we compute the SVD of $L_t^{-1}\Theta_t$ directly, in which case any possible accumulated error is wiped out. In our experiments, we observed that L_t^{-1} converges quickly [7] over t for data consisting of natural signals. In such cases, the exact SVD of $L_t^{-1}\Theta_t$ can be obtained for some initial time instances, after which the approximation (10) is observed to work very well.

An alternative way to perform the transform update (6) would be to use the stochastic gradient descent method. How-

[3] The solution (8) is unique if and only if Θ_t has full rank.

[4] SVD estimates computed at time j are further used in the rank-1 update at time $j+1$.

[5] The factor is $\frac{\max_{2\le j\le t}\|\Theta_j\|_2}{\sigma_n(\Theta_t)}$, where σ_n is the smallest singular value of a matrix, and the $\|\cdot\|_2$ norm denotes the spectral norm. The factor is finite for Θ_t that is full rank.

[6] Since the eigenvectors of a matrix can always be multiplied by -1 to yield equally valid alternative eigenvectors, we may flip the sign of any row of U_j so that the row is a closer match to the corresponding row of U_{j-1} in the bound.

[7] For example, when y_t are independent and identically distributed, $t^{-1}\sum_{j=1}^{t} y_j y_j^T$ converges (as $t\to\infty$) with probability 1 to a covariance matrix, and L_t^{-1} would also converge.

Online Transform Learning Algorithm A1

Input: The sequence $\{y_t\}$.

Initialize: $\hat{W}_0 = W_0$, $\Delta_0 = \Sigma_0 = 0$, $U_0 = Q_0 = R_0 = I$, $\beta_0 = 0$.

For $t = 1, 2, 3, ...$ **Repeat**

1) **Sparse Coding:** $\hat{x}_t = H_s(\hat{W}_{t-1} y_t)$.
2) Update $\beta_t = (1 - t^{-1})\beta_{t-1} + t^{-1} \lambda_0 \left\| y_t \right\|_2^2$.
3) **Transform Update:**
   a) $(U_t, \Delta_t, U_t) \leftarrow \mathcal{U}\left[U_{t-1}, (1 - t^{-1})\Delta_{t-1}, U_{t-1}, A_t\right]$ by rank-1 update, where $A_t = t^{-1} y_t y_t^T$.
   b) $\mathrm{SVD}(L_t^{-1}) \leftarrow (U_t, (\Delta_t + \beta_t I)^{-\frac{1}{2}}, U_t)$.
   c) $(Q_t, \Sigma_t, R_t) \leftarrow \mathcal{U}\left[Q_{t-1}, (1 - t^{-1})\Sigma_{t-1}, R_{t-1}, \tilde{A}_t\right]$ by rank-1 update, where $\tilde{A}_t = t^{-1} L_t^{-1} y_t \hat{x}_t^T$.
   d) Store the scalar/matrix factors for $\hat{W}_t$ in (11), where L_t^{-1} is defined in (b) above.

End

Fig. 1. Algorithm A1 to solve (P1) by alternating minimization [8].

ever, the gradient computation requires computing the inverse of a matrix (a term like W^{-T} [4]). This computation scales worse ($O(n^3)$) than the computation (see Section III-B5) for the proposed method.

While one could alternate multiple times (for each t) between the sparse coding and transform update steps of (P1), we perform only a single alternation to save computations, and to prevent overfitting to the current data. Our overall algorithm for (P1) is shown in Fig. 1.

4) Handling Variations to (P1): Our algorithm can be easily modified to accommodate the various modifications to Problem (P1) suggested in Section II-B for non-stationary data, and for fixed data sets. For example, when making multiple passes over a fixed data set of size N, the update formula (10) would not involve any approximation since $L_t^{-1} = L_{t-1}^{-1} = \hat{L}^{-1}$ after the first pass, where $\hat{L}$ is the square root of $N^{-1} \sum_{j=1}^{N} \left(y_j y_j^T + \lambda_j I\right)$ (the set of training data remains the same when making multiple passes over the data set), and the term $t^{-1} L_t^{-1} y_t \hat{x}_t^T$ in (10) is replaced by $t^{-1} L_t^{-1} y_t (\hat{x}_t - \hat{x}_t')^T$, where $\hat{x}_t'$ is the 'older' version of the sparse code of y_t, which is removed from the formula (and the objective). When the sparse codes are not themselves stored, one can adopt a similar technique as in [47], or use a forgetting factor (1) when making multiple passes over the data set, in order to forget the 'older' bad sparse codes.

When using the forgetting factor ρ (as in (1)) in online transform learning, the various operations for the transform update step of (P1) are modified as follows. We find the SVD square root L_t of $t^{-1} \sum_{j=1}^{t} \rho^{t-j} \left(y_j y_j^T + \lambda_j I\right)$, and compute the full SVD of $t^{-1} \sum_{j=1}^{t} \rho^{t-j} L_t^{-1} y_j \hat{x}_j^T$. The methodology of transform update remains the same as before, except that we work with the modified matrices/scalars $\Gamma_t = \rho(1 - t^{-1})\Gamma_{t-1} + t^{-1} y_t y_t^T$, $\beta_t = \rho(1 - t^{-1})\beta_{t-1} + t^{-1}\lambda_t$, and $\Theta_t = \rho(1 - t^{-1})\Theta_{t-1} + t^{-1} y_t \hat{x}_t^T$, in the aforementioned steps, with $\Gamma_0 = \Theta_0 = 0$ and $\beta_0 = 0$.

[8] If transform update isn't performed for some initial t (Section II-B), then all SVDs are computed exactly for the first transform update. For simplicity, the monitoring of the relative error for (10) is not shown in Fig. 1.

5) Computational and Memory Costs: We now discuss the computational cost and memory requirements of the online transform learning algorithm. The computational cost of the sparse coding step is dominated [4] by the computation of the product $W y_t$, and therefore scales as $O(n^2)$. In contrast, the projection operation in (5) requires only $O(n \log n)$ operations [4], when employing sorting. The computational cost of the transform update step is dominated by $O(n^2 \log^2 n)$ for the rank-1 SVD updates [53], [54]. Thus, the total cost per signal (or, per time instant) of our algorithm (sparse coding and transform update) scales as $O(n^2 \log^2 n)$. This is better (especially for large n) than the computational cost per signal for online learning of an $n \times K$ overcomplete synthesis dictionary D, which scales (assuming synthesis sparsity $s \propto n$, and $K \propto n$) as at least $O(n^3)$ [47]. The (local) memory requirement of our algorithm scales modestly as $O(n^2)$, since we need to store $n \times n$ matrices.

C. Mini-batch Transform Learning

Here, we solve Problem (P2), that processes blocks of signals, by (a single iteration of) alternating minimization. In the sparse coding step, we solve for X_J in (P2), with fixed W ($= \hat{W}_{J-1}$, i.e., warm start) as follows:

$$\min_{X_J} \left\| W Y_J - X_J \right\|_F^2 \quad s.t. \quad \left\| x_{JM-M+i} \right\|_0 \leq s \; \forall \, i \qquad (12)$$

The optimal solution to (12) is obtained as $\hat{x}_{JM-M+i} = H_s(W y_{JM-M+i}) \; \forall \; i \in \{1, .., M\}$.

The transform update step solves (P2) with fixed $X_j = \hat{X}_j$, $1 \leq j \leq J$, as

$$\min_{W} \; \frac{1}{JM} \sum_{j=1}^{J} \left\{ \left\| W Y_j - X_j \right\|_F^2 + \Lambda_j \, Q(W) \right\} \qquad (13)$$

When the block size M is small ($M \ll n$), we can use the same approximate transform update procedure as in Section III-B3, but with the rank-1 updates replaced by rank-M updates. The rank-M updates can be performed as M rank-1 updates for small M. For larger M ($M \sim O(n)$, or larger), (13) is solved using an exact transform update procedure (similar to the one for Problem (6)). Fig. 2 shows the overall algorithm for the case of large M.

We now discuss the computational cost and memory requirements of the mini-batch version of online transform learning. The computational cost of the sparse coding step scales as $O(Mn^2)$. For small M, the cost of the transform update step scales as $O(Mn^2 \log^2 n)$. For large M, since the transform update is performed as in Fig. 2 (i.e., matrix inverses, matrix-matrix multiplications, and SVDs are computed directly, but scalars/matrices are accumulated over time wherever possible), the cost of transform update (Steps 2 and 3 in Fig. 2) scales as $C_1 Mn^2 + C_2 n^3$, where C_1 and C_2 are constants. Assuming that $C_2 n < C_1 M$ (large M), the transform update cost scales as $O(Mn^2)$. Thus, the total computation per iteration (or, per block) of our mini-batch algorithm (sparse coding and transform update) scales as $O(Mn^2)$ for large M, and $O(Mn^2 \log^2 n)$ for small M. In each of these cases, the cost is better than the cost per block (of size M) for mini-batch learning of an $n \times K$ synthesis dictionary D, which

Mini-batch Transform Learning Algorithm A2

Input: Sequence $\{y_t\}$ is processed in blocks of size M.

Initialize: $\hat{W}_0 = W_0$, $\tilde{\Theta}_0 = \tilde{\Gamma}_0 = 0$, $\tilde{\beta}_0 = 0$.

For $J = 1, 2, 3, ...$ **Repeat**

1) **Sparse Coding:** $\hat{x}_l = H_s(\hat{W}_{J-1}\ y_l)\ \forall\ l$ such that $JM - M + 1 \leq l \leq JM$.
2) **Prepare for Transform Update:**

$$Y_J = \left[y_{JM-M+1} \mid y_{JM-M+2} \mid \quad ... \quad \mid y_{JM}\right].$$

$$\hat{X}_J = \left[\hat{x}_{JM-M+1} \mid \hat{x}_{JM-M+2} \mid \quad ... \quad \mid \hat{x}_{JM}\right].$$

$$\tilde{\Theta}_J = (1 - J^{-1})\,\tilde{\Theta}_{J-1} + J^{-1}M^{-1}Y_J\hat{X}_J^T.$$

$$\tilde{\Gamma}_J = (1 - J^{-1})\,\tilde{\Gamma}_{J-1} + J^{-1}M^{-1}Y_JY_J^T.$$

$$\tilde{\beta}_J = (1 - J^{-1})\,\tilde{\beta}_{J-1} + J^{-1}M^{-1}\lambda_0\left\|Y_J\right\|_F^2.$$

3) **Transform Update:**
   a) Compute the full SVD of $\tilde{\Gamma}_J + \tilde{\beta}_J I$ as $\tilde{U}_J\tilde{\Delta}_J\tilde{U}_J^T$.
   b) $\tilde{L}_J^{-1} = \tilde{U}_J\tilde{\Delta}_J^{-\frac{1}{2}}\tilde{U}_J^T$ (inverse square root).
   c) Compute full SVD of $\tilde{L}_J^{-1}\tilde{\Theta}_J$ as $\tilde{Q}_J\tilde{\Sigma}_J\tilde{R}_J^T$.
   d) $\hat{W}_J = 0.5\tilde{R}_J\left(\tilde{\Sigma}_J + \left(\tilde{\Sigma}_J^2 + 2\tilde{\beta}_J I\right)^{\frac{1}{2}}\right)\tilde{Q}_J^T\tilde{L}_J^{-1}$.

End

Fig. 2. Algorithm A2 to solve (P2) for large block size M.

scales (assuming synthesis sparsity $s \propto n$, and $K \propto n$) as $O(Mn^3)$ [47]. The memory requirement of mini-batch transform learning scales as $O(Mn)$ for large M, and $O(n^2)$ for small M.

D. Comparison of Transform Learning Schemes

We now compare and contrast the online, mini-batch, and batch transform learning schemes in terms of their computational costs, memory requirements, and latency. We measure latency as the time duration (the inter-arrival time between two signals is taken as 1 time unit) between the arrival of the first signal, and the generation of its corresponding output (e.g., sparse code) [9].

Table I summarizes the various costs for the transform-based schemes. We show the computational cost per sample, i.e., the total cost normalized by the number of samples processed. For a given number of N samples, the batch scheme typically requires several iterations, their number denoted by P, to converge to a good transform. Thus, the batch scheme has a total computational cost of $O(PNn^2)$. In practice, P depends on n, N, and algorithm initialization, and is typically larger for bigger, or more complex problems. On the other hand, as shown in Part II [45], and in the experiments of Section IV, the online and mini-batch schemes produce good transforms for N (total number of signals processed sequentially) large. Therefore, the net computational cost for processing N signals (and converging) for the online scheme is $O(Nn^2\log^2 n)$. Thus, assuming $\log^2 n < P$ (which is typically observed in practice), the online scheme is computationally more effective

[9]Here, for simplicity, we assume that computations can be performed instantaneously.

Properties	Online	Mini-batch		Batch
		Small M	Large M	
Computations	$O(n^2\log^2 n)$	$O(n^2\log^2 n)$	$O(n^2)$	$O(Pn^2)$
Memory	$O(n^2)$	$O(n^2)$	$O(nM)$	$O(nN)$
Latency	0	$M-1$	$M-1$	$N-1$

TABLE I
COMPARISON OF ONLINE LEARNING, MINI-BATCH LEARNING, AND BATCH LEARNING IN TERMS OF THEIR COMPUTATIONAL COST PER SAMPLE, MEMORY COST, AND LATENCY.

(in order) than the batch scheme for big data (large N). The computational cost of processing N signals (and thus converging, in the case of large N) for the mini-batch scheme for large M (and N/M blocks) is $O(Nn^2)$, which is even lower in order (by factor $\log^2 n$) than the cost for the (one signal at a time) online scheme.

Importantly, assuming $n, M \ll N$, the online and mini-batch transform learning schemes have far lower memory requirements and latency compared to the batch scheme. The mini-batch scheme itself has higher memory and latency costs than the online scheme.

As discussed in the preceding subsections, the dictionary learning schemes [5], [47] have computational cost per sample (not shown in Table I) proportional to n^3. For large signals (i.e., large n), the cost of dictionary learning is much larger than the cost of the transform-based methods.

E. Denoising

Problem (P3) is identical to (P1), except for the fact that it uses a sparsity penalty function, rather than constraints. Therefore, when solving (P3) at each t by alternating minimization, the sparse coding step (with fixed $W = \hat{W}_{t-1}$) is

$$\min_{x_t}\ \|Wy_t - x_t\|_2^2 + \tau_t^2\left\|x_t\right\|_0 \tag{14}$$

A solution [46] $\hat{x}_t$ of (14) is $\hat{x}_t = \hat{H}_{\tau_t}(Wy_t)$, where the hard thresholding operator $\hat{H}_\tau(\cdot)$ is defined as

$$\left(\hat{H}_\tau(b)\right)_k = \begin{cases} 0 & ,\ |b_k| < \tau \\ b_k & ,\ |b_k| \geq \tau \end{cases} \tag{15}$$

where $b \in \mathbb{R}^n$, and the subscript k indexes vector entries. Therefore, the sparse coding solution is simply obtained by hard thresholding, with a threshold proportional to the noise level σ (similar to traditional techniques involving analytical transforms [55]). The transform update step of (P3) is identical to (P1). The denoised signal is computed as $\hat{W}_t^{-1}\hat{x}_t$. By, (11), we have

$$\hat{W}_t^{-1} = \beta_t^{-1}L_tQ_t\left(\left(\Sigma_t^2 + 2\beta_t I\right)^{\frac{1}{2}} - \Sigma_t\right)R_t^T \tag{16}$$

Assuming that the various matrices (or their EVDs) in the above decomposition are stored in memory, $\hat{W}_t^{-1}\hat{x}_t$ can be computed using matrix-vector multiplications at a cost of $O(n^2)$. The net computational cost of denoising per signal (i.e. for sparse coding, transform update, and denoised signal computation) then scales as $O(n^2\log^2 n)$ (same cost as for the online learning algorithm for (P1)). For mini-batch transform learning based denoising, the net computational cost of denoising a block is similar to the costs mentioned in Section III-C.

IV. Numerical Experiments

A. Framework

In Part II [45], it is shown that online transform learning converges asymptotically, and produces a good transform. Here, we present numerical results illustrating the practical convergence behavior of online (and mini-batch) transform learning, as well as the usefulness of the proposed schemes for image representation and denoising. First, we consider synthetic data generated sequentially using a particular transform model, and study the ability of our online schemes to converge to a good model. Second, we study the usefulness of online/sequential transform learning for sparse representation of images. Finally, we present results for online denoising using Problem (P3). We consider the patch-based denoising of some standard (regular sized) images as well as some very large images (where batch learning was observed to be infeasible on the particular computing platform used for our experiment). The latter case is a candidate big data problem, since it involves a large number of patches, which can be potentially denoised efficiently and sequentially using online transform learning.

All transform learning implementations were coded in Matlab version R2013b. Similarly, the Matlab implementation of K-SVD denoising [6] (a popular batch synthesis dictionary-based denoising scheme) available from Michael Elad's website [56] was used in our comparisons. For K-SVD denoising, we used the built-in parameter settings of the author's implementation. All computations were performed with an Intel Core i7 CPU at 2.9GHz and 4GB memory, employing a 64-bit Windows 7 operating system.

We define the normalized reconstruction error as $\left\|Y - W^{-1}X\right\|_F^2 / \ \|Y\|_F^2$, where Y is a matrix whose columns are the data vectors, W is a transform, and X is the corresponding sparse code matrix. The normalized reconstruction error metric is used to measure the performance of learnt transforms for signal/image representation. It can be thought of as a simple surrogate for the compression performance of a transform. For image denoising, similar to prior work, we measure the peak-signal-to-noise ratio (PSNR) computed between the true noiseless reference, and the noisy or denoised images.

B. Convergence and Sparse Representation

1) Convergence: First, we illustrate the convergence behavior of the proposed algorithms. We generate the input data y_t sequentially as $W^{-1}x_t$ using a random orthonormal 20×20 matrix W, and sparse codes x_t obtained by thresholding i.i.d. Gaussian vectors at sparsity level $s = 3$. We then use our online and mini-batch transform learning algorithms for (P1) and (P2), to sequentially learn the transform and sparse codes for the data y_t. The parameter $\lambda_0 = 3.1 \times 10^{-2}$, $s = 3$, and the size of the mini-batch $M = 320$. We test (and compare) both the exact (see Section III-B2) and approximate (see Section III-B3) versions of the online transform learning algorithm. As discussed in Section III-B3, we monitor the upper bound on the relative approximation error. If it rises above a threshold $\epsilon = 10$, we compute the SVD of $L_t^{-1}\Theta_t$ directly in the transform update step of the algorithm. Our algorithms are initialized with the 20×20 DCT matrix.

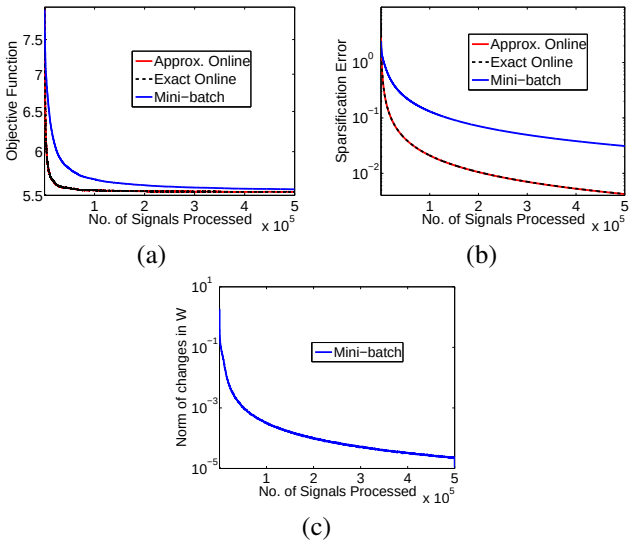


Fig. 3. Convergence behavior of the online (both the exact and approximate versions) and mini-batch learning schemes as a function of amount of data processed: (a) Objective function, (b) Sparsification error, (c) $||\hat{W}_{t+1} - \hat{W}_t||_F$ for mini-batch scheme.

Figs. 3(a) and 3(b) show the objective functions of Problems (P1), (P2), and sparsification errors (i.e., the objectives without the regularizers) as a function of the number of signals processed, for our online and mini-batch schemes. Both the objective and sparsification error converge quickly for our schemes. The exact and approximate online schemes behave identically. Moreover, for the approximate version, we observed that the error threshold ϵ is violated (i.e., an exact SVD is performed) only 0.2% of the time. This indicates that the faster approximate online scheme works equally well as the exact version. The online schemes converge slightly faster than the mini-batch scheme as a function of the number of signals processed. This is because the transform update step is performed more frequently for the (one signal at a time) online schemes. However, the proposed approximate online transform learning scheme typically has a higher run time (due to the $\log^2 n$ factor in computations – see Section III-D) than the mini-batch scheme.

As shown in Fig. 3(c), the difference between successive iterates, i.e., $\left\|\hat{W}_{t+1} - \hat{W}_t\right\|_F$ converges close to zero for the mini-batch scheme. A similar behavior is observed for the online schemes. The learned transforms using our exact online, approximate online, and mini-batch algorithms have condition numbers of 1.02, 1.02, and 1.04 respectively. By Fig. 3(b), they provide a sparsification error close to zero. The normalized reconstruction error computed using the learned $\hat{W}_t$ and the sparse codes generated sequentially is < 0.01 for our schemes, indicating that they have learned a good model for the data $\{y_t\}$.

2) Sparse Representation of Images: Here, we learn a sparsifying transform for natural image patches. We extract all non-overlapping patches of size 8×8 (about 1.6×10^5 patches) from the images in the USC-SIPI database [57] (the

	Batch	Mini-batch		Online	
		1 pass	30 passes	1 pass	30 passes
Reconstruction Error Improv.	1.3	0.8	1.3	0.8	1.3

TABLE II
RECONSTRUCTION ERROR IMPROVEMENTS (dB) OVER PATCH-BASED 2D DCT FOR THE BATCH, MINI-BATCH, AND ONLINE TRANSFORM LEARNING SCHEMES FOR IMAGE DATA. THE RESULTS FOR THE MINI-BATCH AND ONLINE SCHEMES ARE ALSO SHOWN WITH MULTIPLE PASSES THROUGH THE DATASET.

Images	σ	Noisy PSNR		Batch K-SVD	Batch TL	Mini-batch TL
Couple	5	34.16	PSNR	37.28	37.33	37.33
			time	1250	92	20
	10	28.11	PSNR	33.51	33.62	33.62
			time	671	68	19
	20	22.11	PSNR	30.02	30.02	30.03
			time	190	61	20
Man	5	34.15	PSNR	36.47	36.66	36.75
			time	1279	205	45
	10	28.13	PSNR	32.71	32.96	33.00
			time	701	130	44
	20	22.11	PSNR	29.40	29.57	29.52
			time	189	80	41

TABLE III
PSNR VALUES (dB) AND RUN TIMES (SECONDS) FOR DENOISING REGULAR-SIZE IMAGES AT DIFFERENT NOISE LEVELS ($\sigma = 5$, 10, 20), USING BATCH K-SVD [6], BATCH TRANSFORM LEARNING [9], AND MINI-BATCH TRANSFORM LEARNING-BASED DENOISING SCHEMES.

color images are converted to gray-scale images). We use our (approximate) online transform learning algorithm, and the mini-batch learning algorithm to learn a transform and sparse codes sequentially on the (mean-subtracted) patches. The parameters are set as $\lambda_0 = 6.2 \times 10^{-2}$, $s = 11$, and $M = 256$.

Table II shows the improvements in the normalized reconstruction error achieved by the online, mini-batch and batch [4] transform learning schemes over the fixed 2D DCT for the image patches. For the online and mini-batch schemes, we also show results with multiple passes through the same data (each time a particular signal is repeated, its old sparse code is replaced with the latest one – see Section III-B4). Both the online and mini-batch schemes (with either a single pass through the data, or with multiple passes) provide better reconstruction quality compared to the DCT. The proposed schemes also perform similar to the batch scheme (and about 1.3 dB better than the DCT) at the end of 30 passes. Importantly, even with 30 passes, the mini-batch scheme runs $3\times$ faster than the batch algorithm in achieving similar reconstruction quality. The learned transforms were all well-conditioned in this experiment.

C. Online Image Denoising

1) Regular-size Image Denoising: Here, we present some results for our simple denoising framework (P3). We consider image denoising, where the overlapping image patches are processed and denoised sequentially. We work with the images Couple (512×512) and Man (768×768) [10], and simulate i.i.d. additive Gaussian noise at three different noise levels (standard deviation $\sigma = 5$, 10, 20) for the images.

We denoise the 8×8 overlapping image patches (the mean is subtracted during learning, and added back in the reconstruction step) sequentially (we make only one pass through the set of patches) using adaptive mini-batch denoising. Once the denoised patches in each mini-batch are computed, we immediately put them back at their corresponding locations in the denoised image. Note that the denoised image is computed by averaging the denoised patches at their respective 2D locations. Our scheme requires minimal memory (we only store data required for computations at a particular time instant t) and mimics a real-time denoising setup.

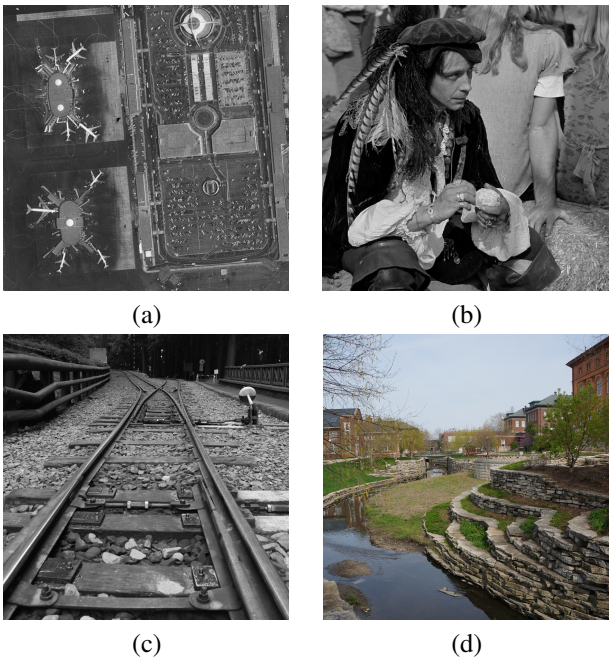
The parameters are set to $\lambda_0 = 3.1 \times 10^{-2}$, $M = 64$, and $\tau_j = 1.73\sigma \ \forall j$. We work with a forgetting factor $\rho < 1$ (corresponding to an extension of (1) for the mini-batch case), which was found to provide a slight improvement. The best choice of ρ depends on the size of the mini-batch, patch size (signal size), and noise level, and was set empirically to $\rho = 0.87$, 0.95, and 0.99, for $\sigma = 5$, 10, and 20, respectively.

Our denoising results are compared to K-SVD denoising [5], [6], [56], and to batch square transform learning-based denoising [9] (with parameters set as in [40]). The images in this experiment have sizes compatible (i.e., small enough to avoid memory overflows) with the batch denoising schemes. The goal of the comparison to the batch dictionary/transform schemes is not to show the state-of-the-art performance of our method in a general denoising application. Rather, we focus on the sequential aspect of our method, and aim to demonstrate that the proposed mini-batch transform learning-based denoising algorithm with smaller latency, memory and computational requirements, can be used as an efficient and effective alternative to adaptive batch mode dictionary/transform denoising.

Table III lists the denoising PSNRs and run times (including the time for generating the denoised image) for the various methods. The mini-batch transform learning-based denoising method provides comparable, or better denoising performance compared to the batch-based methods, while being much faster. We compute the average speedup provided by our mini-batch denoising scheme over the adaptive batch-based methods. For each image and noise level, the ratio of the run times of batch denoising and mini-batch denoising is computed, and this speedup is averaged over the images and noise levels in Table III. The mini-batch transform learning-based denoising scheme provides an average speedup of $26.0\times$ and $3.4\times$ respectively, over the batch K-SVD and batch transform denoising schemes.

2) Large-Scale Image Denoising: In the context of big data, batch learning is typically infeasible due to the strict practical limits on computational and memory resources. However, we can potentially use the proposed online, or mini-batch transform learning schemes to denoise large images, and image sequences. Here, we present preliminary results for patch-

[10] These are standard images that have been used in prior work (e.g., [6], [40]).



(a) (b) (c) (d)

Fig. 4. The large images used in our denoising experiments: (a) Airport (1024×1024), (b) Man (1024×1024), (c) Railway (2048×2048), (d) Campus ($3264 \times 3264 \times 3$).

based mini-batch denoising of large-scale images (both gray-scale and color images). We work with the gray-scale images in Fig. 4(a)-(c), and the color image in Fig. 4(d). The largest of these images has about 11 Megapixels (when it is processed by a patch-based scheme, there are about 11 million overlapping patches in total). We simulate i.i.d. additive Gaussian noise at 3 different noise levels ($\sigma = 20,\ 50,\ 100$) for these images. In the case of the color image, noise is added to each of the red (R), green (G), and blue (B) color channels.

The size of the mini-batch is set to $M = 256$. The forgetting factor is set to $\rho = 0.88,\ 0.92$, and 0.95, for $\sigma = 20,\ 50$, and 100 respectively, for gray-scale images, and $\rho = 0.95$ for the color image. Other parameters in our algorithm are set as in the regular-size image denoising experiment described earlier. To apply the transform, we vectorize all image patches. For color image denoising, we form a patch vector by stacking the patch's vectorized red, green, and blue components [33]. We therefore learn a larger 192×192 transform for the patches of the color image.

Table IV lists the denoising PSNRs and run times (the latter are averaged over the noise levels σ for each image) obtained using the proposed simple mini-batch transform learning-based denoising algorithm, as well as with a similar scheme but involving the fixed patch-based 2D DCT (no learning). For the DCT-based algorithm, we use $\tau_j = 2.45\sigma\ \forall j$, which was found to be empirically optimal in our experiments. As evidenced from Table IV, the mini-batch denoising algorithm provides better PSNRs than the DCT for all images and noise levels [11]. To denoise the large-scale images, the mini-batch algorithm takes up to 23% longer for 1024×1024 images (with rougly 1 million patches) or for the 2048×2048 image (roughly 4 million patches), and about 40% longer for the 3264×3264 color image (roughly 11 million patches) respectively, than the 2D DCT method. Therefore, although there is no learning involved in the latter case, our adaptive scheme typically denoises about as fast as the fixed transform.

[11] We did not observe any marked improvement in the PSNRs, when replacing DCT with other fixed transforms, such as Wavelets.

Images	Methods	$\sigma = 20$ (22.11)	$\sigma = 50$ (14.15)	$\sigma = 100$ (8.13)	Run Times
Airport	DCT	28.79	24.65	21.00	23
	TL	**28.83**	**25.07**	**22.53**	28
Man	DCT	30.44	25.80	21.87	23
	TL	**30.64**	**26.62**	**23.88**	27
Railway	DCT	31.90	26.44	22.04	90
	TL	**32.42**	**27.58**	**24.35**	111
Campus	DCT	30.89	25.88	21.99	323
	TL	**33.10**	**27.47**	**23.24**	451

TABLE IV

PSNR VALUES (dB) AND RUN TIMES (SECONDS) FOR DENOISING THE LARGE GRAY-SCALE AND COLOR IMAGES WITH MINI-BATCH TRANSFORM LEARNING (TL) BASED METHOD, AND THE 2D DCT AT DIFFERENT NOISE LEVELS ($\sigma = 20,\ 50,\ 100$). THE NOISY IMAGE PSNRS ARE GIVEN UNDER NOISE LEVEL σ IN PARENTHESES. THE BEST DENOISING PSNR FOR EACH NOISE LEVEL AND IMAGE IS MARKED IN BOLD.

(a) (b)

Fig. 5. Zoom-in of large-scale image (Man 1024×1024) denoising results: (a) Noisy image (PSNR = $22.11\,dB$), (b) Denoised image using the proposed adaptive mini-batch scheme (PSNR = $30.64\,dB$).

Figure 5 shows a zoom-in of the denoised Man image, obtained using the proposed mini-batch transform learning-based denoising algorithm at $\sigma = 20$. The zoom-in shows reasonable reconstruction of the image features from the noisy measurements. Thus, the results here demonstrate the potential of our scheme for limited latency (or, real-time) denoising of large-scale data.

A feasible alternative method for large-scale image denoising is to break the large image into smaller "mini-images", and perform batch denoising on each of the mini-images. To test the effectiveness of this alternative, we divided each of our large noisy images into 256×256 overlapping (an overlap of 7 pixels) mini-images, and performed batch transform denoising on the mini-images [9]. The mini-images were denoised one-by-one. This allows us to use the transform learnt by the batch denoising algorithm in the current mini-image as an initialization (for the first mini-image, the initialization used is the 2D DCT) for the batch denoising scheme [9] in the next nearest mini-image. Once all the mini-images are denoised by the batch method, they are averaged together at their

corresponding 2D locations in the large image. We observed that this alternative denoising scheme performs (on the noisy images in Table IV) comparably, or worse (by $0.1 - 0.3$ dB) compared to our proposed mini-batch transform learning-based denoising scheme. Importantly, the alternative mini-image-based batch method is $7\times$ slower on the average.

As we have emphasized, the proposed adaptive online and mini-batch schemes are capable of being applied to realistic tasks such as real-time sparse coding (compression), and denoising. The idea of image inpainting using mini-batch synthesis dictionary learning has been discussed in [47]. However, the scheme therein does not solve a real-time adaptive inpainting problem. Rather, a dictionary is first learnt over all the data, and then later used to reconstruct (with fixed dictionary) the patches. Therefore, we do not directly compare to that work here.

V. Conclusions

In this paper, we presented a novel problem formulation for online learning of square sparsifying transforms. The formulation is to sequentially update the sparsifying transform and sparse code for signals that arrive or, are processed sequentially. The proposed algorithm involves a sparse coding step and a transform update step per signal. Each of these steps is implemented efficiently. We also presented a mini-batch version of our online algorithm that can handle blocks of signals at a time. The proposed schemes were shown to be computationally much cheaper (in terms of cost per signal) than online synthesis dictionary learning. In practice, the online/mini-batch sparsifying transform learning converges quickly. We presented experiments demonstrating the usefulness of online transform learning in sparse signal representation, and denoising. The topics of online learning of an overcomplete transform, and online video denoising will be considered in future work.

Appendix A
Approximation Error in Online Algorithm

Here, we calculate the error introduced by the approximation (10) in the transform update step of our online learning algorithm. Let us denote $L_t^{-1}\Theta_t$, $t^{-1}L_t^{-1}y_t x_t^T$, and $\left(L_t^{-1} - L_{t-1}^{-1}\right)\Theta_{t-1}$ by M_t, z_t, and Υ_t, respectively. Then, by (9), we have

$$M_t = (1 - t^{-1})M_{t-1} + (1 - t^{-1})\Upsilon_t + z_t \qquad (17)$$

Equation (10) introduces an approximation to M_t, and then computes the SVD of the approximate matrix. Let us denote the approximate matrix as $\hat{M}_t$. The SVD of $\hat{M}_t$ is computed via a rank-1 update to the SVD of $(1 - t^{-1})\hat{M}_{t-1}$. Thus, we have by (10) that

$$\hat{M}_t = (1 - t^{-1})\hat{M}_{t-1} + z_t \qquad (18)$$

Subtracting (18) from (17) and denoting $M_t - \hat{M}_t$ by E_t, yields

$$E_t = (1 - t^{-1})(E_{t-1} + \Upsilon_t) \qquad (19)$$

Assuming $E_1 = 0$, equation (19) implies that $E_t = \sum_{j=2}^{t} \frac{j-1}{t}\Upsilon_j$. ■

References

[1] M. Elad, P. Milanfar, and R. Rubinstein, "Analysis versus synthesis in signal priors," *Inverse Problems*, vol. 23, no. 3, pp. 947–968, 2007.

[2] S. Nam, M. E. Davies, M. Elad, and R. Gribonval, "Cosparse analysis modeling - uniqueness and algorithms," in *ICASSP*, 2011, pp. 5804–5807.

[3] W. K. Pratt, J. Kane, and H. C. Andrews, "Hadamard transform image coding," *Proc. IEEE*, vol. 57, no. 1, pp. 58–68, 1969.

[4] S. Ravishankar and Y. Bresler, "Learning sparsifying transforms," *IEEE Trans. Signal Process.*, vol. 61, no. 5, pp. 1072–1086, 2013.

[5] M. Aharon, M. Elad, and A. Bruckstein, "K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation," *IEEE Trans. Signal Process.*, vol. 54, no. 11, pp. 4311–4322, 2006.

[6] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," *IEEE Trans. Image Process.*, vol. 15, no. 12, pp. 3736–3745, 2006.

[7] M. Yaghoobi, S. Nam, R. Gribonval, and M. E. Davies, "Constrained overcomplete analysis operator learning for cosparse signal modelling," *IEEE Trans. Signal Process.*, vol. 61, no. 9, pp. 2341–2355, 2013.

[8] R. Rubinstein, T. Peleg, and M. Elad, "Analysis K-SVD: A dictionary-learning algorithm for the analysis sparse model," *IEEE Transactions on Signal Processing*, vol. 61, no. 3, pp. 661–677, 2013.

[9] S. Ravishankar and Y. Bresler, "Closed-form solutions within sparsifying transform learning," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013, pp. 5378–5382.

[10] A. M. Bruckstein, D. L. Donoho, and M. Elad, "From sparse solutions of systems of equations to sparse modeling of signals and images," *SIAM Review*, vol. 51, no. 1, pp. 34–81, 2009.

[11] B. K. Natarajan, "Sparse approximate solutions to linear systems," *SIAM J. Comput.*, vol. 24, no. 2, pp. 227–234, Apr. 1995.

[12] G. Davis, S. Mallat, and M. Avellaneda, "Adaptive greedy approximations," *Journal of Constructive Approximation*, vol. 13, no. 1, pp. 57–98, 1997.

[13] Y. Pati, R. Rezaiifar, and P. Krishnaprasad, "Orthogonal matching pursuit : recursive function approximation with applications to wavelet decomposition," in *Asilomar Conf. on Signals, Systems and Comput.*, 1993, pp. 40–44 vol.1.

[14] S. G. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *IEEE Trans. Signal Process.*, vol. 41, no. 12, pp. 3397–3415, 1993.

[15] W. Dai and O. Milenkovic, "Subspace pursuit for compressive sensing signal reconstruction," *IEEE Trans. Information Theory*, vol. 55, no. 5, pp. 2230–2249, 2009.

[16] S. S. Chen, D. L. Donoho, and M. A. Saunders, "Atomic decomposition by basis pursuit," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 33–61, 1998.

[17] I. F. Gorodnitsky, J. George, and B. D. Rao, "Neuromagnetic source imaging with FOCUSS: A recursive weighted minimum norm algorithm," *Electrocephalography and Clinical Neurophysiology*, vol. 95, pp. 231–251, 1995.

[18] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least angle regression," *Annals of Statistics*, vol. 32, pp. 407–499, 2004.

[19] G. Harikumar, "Blind image deconvolution from multiple blurs, and sparse approximation," Ph.D. dissertation, University of Illinois at Urbana-Champaign, mar 1997, Yoram Bresler, adviser.

[20] R. Chartrand, "Exact reconstruction of sparse signals via nonconvex minimization," *Signal Processing Letters, IEEE*, vol. 14, no. 10, pp. 707–710, 2007.

[21] D. Donoho, "Compressed sensing," *IEEE Trans. Information Theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[22] J. A. Tropp, "Greed is good: Algorithmic results for sparse approximation," *IEEE Trans. Inform. Theory*, vol. 50, no. 10, pp. 2231–2242, 2004.

[23] E. Candès and T. Tao, "Decoding by linear programming," *IEEE Trans. on Information Theory*, vol. 51, no. 12, pp. 4203–4215, 2005.

[24] D. L. Donoho, "For most large underdetermined systems of linear equations the minimal l1-norm solution is also the sparsest solution," *Comm. Pure Appl. Math*, vol. 59, pp. 797–829, 2004.

[25] R. Rubinstein, T. Faktor, and M. Elad, "K-SVD dictionary-learning for the analysis sparse model," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 5405–5408.

[26] R. Rubinstein and M. Elad, "K-SVD dictionary-learning for analysis sparse models," in *Proc. SPARS11*, June 2011, p. 73.

[27] A. Chambolle, "An algorithm for total variation minimization and applications," *J. Math. Imaging Vis.*, vol. 20, no. 1-2, pp. 89–97, 2004.

[28] M. Yaghoobi, S. Nam, R. Gribonval, and M. E. Davies, "Noise aware analysis operator learning for approximately cosparse signals," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 5409–5412.

[29] S. Mallat, *A Wavelet Tour of Signal Processing*. San Diego, CA: Academic Press, 1999.

[30] E. J. Candès and D. L. Donoho, "Curvelets - a surprisingly effective nonadaptive representation for objects with edges," in *Curves and Surfaces*. Vanderbilt University Press, 1999, pp. 105–120.

[31] B. A. Olshausen and D. J. Field, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, no. 6583, pp. 607–609, 1996.

[32] K. Engan, S. O. Aase, and J. H. Husoy, "Method of optimal directions for frame design," in *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1999, pp. 2443–2446.

[33] J. Mairal, M. Elad, and G. Sapiro, "Sparse representation for color image restoration," *IEEE Trans. on Image Processing*, vol. 17, no. 1, pp. 53–69, 2008.

[34] M. Yaghoobi, T. Blumensath, and M. Davies, "Dictionary learning for sparse approximations with the majorization method," *IEEE Transaction on Signal Processing*, vol. 57, no. 6, pp. 2178–2191, 2009.

[35] S. Ravishankar and Y. Bresler, "MR image reconstruction from highly undersampled k-space data by dictionary learning," *IEEE Trans. Med. Imag.*, vol. 30, no. 5, pp. 1028–1041, 2011.

[36] K. Skretting and K. Engan, "Recursive least squares dictionary learning algorithm," *IEEE Transactions on Signal Processing*, vol. 58, no. 4, pp. 2121–2130, 2010.

[37] G. Peyré and J. Fadili, "Learning analysis sparsity priors," in *Proc. Int. Conf. Sampling Theory Appl. (SampTA)*, Singapore, May 2-6, 2011. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00542016/

[38] M. Yaghoobi, S. Nam, R. Gribonval, and M. Davies, "Analysis operator learning for overcomplete cosparse representations," in *European Signal Processing Conference (EUSIPCO)*, 2011, pp. 1470–1474.

[39] B. Ophir, M. Elad, N. Bertin, and M. Plumbley, "Sequential minimal eigenvalues - an approach to analysis dictionary learning," in *Proc. European Signal Processing Conference (EUSIPCO)*, 2011, pp. 1465–1469.

[40] S. Ravishankar and Y. Bresler, "Learning doubly sparse transforms for images," *IEEE Trans. Image Process.*, vol. 22, no. 12, pp. 4598–4612, 2013.

[41] ——, "Sparsifying transform learning for compressed sensing MRI," in *Proc. IEEE Int. Symp. Biomed. Imag.*, 2013, pp. 17–20.

[42] B. Wen, S. Ravishankar, and Y. Bresler, "Structured overcomplete sparsifying transform learning with convergence guarantees and applications," *International Journal of Computer Vision*, pp. 1–31, 2014. [Online]. Available: http://dx.doi.org/10.1007/s11263-014-0761-1

[43] L. Pfister, "Tomographic reconstruction with adaptive sparsifying transforms," Master's thesis, University of Illinois at Urbana-Champaign, Aug. 2013.

[44] L. Pfister and Y. Bresler, "Model-based iterative tomographic reconstruction with adaptive sparsifying transforms," in *SPIE International Symposium on Electronic Imaging: Computational Imaging XII*, vol. 9020, 2014, pp. 90 200H–1–90 200H–11.

[45] S. Ravishankar and Y. Bresler, "Online sparsifying transform learning - part II: Convergence analysis," *IEEE Journal of Selected Topics in Signal Process.*, 2015, to appear.

[46] ——, "ℓ_0 sparsifying transform learning with efficient optimal updates and convergence guarantees," *IEEE Trans. Signal Process.*, 2015, to appear.

[47] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online learning for matrix factorization and sparse coding," *J. Mach. Learn. Res.*, vol. 11, pp. 19–60, 2010.

[48] C. Lu, J. Shi, and J. Jia, "Online robust dictionary learning," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013, pp. 415–422.

[49] J. Xing, J. Gao, B. Li, W. Hu, and S. Yan, "Robust object tracking with online multi-lifespan dictionary learning," in *IEEE International Conference on Computer Vision (ICCV)*, Dec 2013, pp. 665–672.

[50] G. Zhang, Z. Jiang, and L. S. Davis, "Online semi-supervised discriminative dictionary learning for sparse representation," in *Computer Vision ACCV 2012*, 2013, pp. 259–273.

[51] L. Wang, K. Lu, P. Liu, R. Ranjan, and L. Chen, "Ik-svd: Dictionary learning for spatial big data via incremental atom update," *Computing in Science & Engineering*, vol. 16, no. 4, pp. 41–52, 2014.

[52] S. Mukherjee and C. S. Seelamantula, "A split-and-merge dictionary learning algorithm for sparse representation," 2014, preprint: http://arxiv.org/abs/1403.4781.

[53] P. Stange, "On the efficient update of the singular value decomposition," *PAMM*, vol. 8, no. 1, pp. 10 827–10 828, 2008.

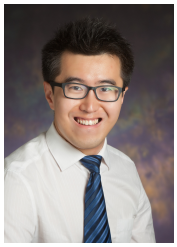
[54] ——, "On the efficient update of the singular value decomposition subject to rank-one modifications," 2011, online: http://www.digibib.tu-bs.de/?docid=00040371.

[55] D. L. Donoho and I. M. Johnstone, "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, vol. 81, no. 3, pp. 425–455, 1994.

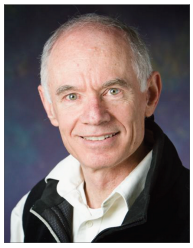
[56] M. Elad, "Michael Elad personal page," http://www.cs.technion.ac.il/~elad/Various/KSVD_Matlab_ToolBox.zip, [Online; accessed 2014].

[57] "The USC-SIPI Image Database," http://sipi.usc.edu/database/database.php?volume=misc, [Online; accessed July-2014].

Saiprasad Ravishankar received the B.Tech. degree in Electrical Engineering from the Indian Institute of Technology Madras, in 2008. He received the M.S. and Ph.D. degrees in Electrical and Computer Engineering, in 2010 and 2014 respectively, from the University of Illinois at Urbana-Champaign, where he is currently an Adjunct Lecturer at the Department of Electrical and Computer Engineering, and a postdoctoral research associate at the Coordinated Science Laboratory. His current research interests include signal and image processing, medical imaging, inverse problems, image analysis, dictionary learning, compressed sensing, machine learning, computer vision, and big data applications.



Bihan Wen received the B.Eng. degree from the Department of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, in 2012. He is currently pursuing the M.S. and Ph.D. degrees in Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign, Urbana, IL, USA. His current research interests include machine learning, sparse coding, image and video enhancement, denoising, and big data applications.



Yoram Bresler received the B.Sc. (cum laude) and M.Sc. degrees from the Technion, Israel Institute of Technology, in 1974 and 1981 respectively, and the Ph.D degree from Stanford University, in 1986, all in Electrical Engineering. In 1987 he joined the University of Illinois at Urbana-Champaign, where he is currently a Professor at the Departments of Electrical and Computer Engineering and Bioengineering, and at the Coordinated Science Laboratory. Yoram Bresler is also President and Chief Technology Officer at InstaRecon, Inc., a startup he co-founded to commercialize breakthrough technology for tomographic reconstruction developed in his academic research. His current research interests include multi-dimensional and statistical signal processing and their applications to inverse problems in imaging, and in particular compressed sensing, computed tomography, and magnetic resonance imaging.

Dr. Bresler has served on the editorial board of a number of journals including the IEEE Transactions on Signal Processing, the IEEE Journal on Selected Topics in Signal Processing, Machine Vision and Applications, and the SIAM Journal on Imaging Science, and on various committees of the IEEE. Dr. Bresler is a fellow of the IEEE and of the AIMBE. He received two Senior Paper Awards from the IEEE Signal Processing society, and a paper he coauthored with one of his students received the Young Author Award from the same society in 2002. He is the recipient of a 1991 NSF Presidential Young Investigator Award, the Technion (Israel Inst. of Technology) Fellowship in 1995, and the Xerox Senior Award for Faculty Research in 1998. He was named a University of Illinois Scholar in 1999, appointed as an Associate at the Center for Advanced Study of the University in 2001-02, and Faculty Fellow at the National Center for Super Computing Applications in 2006.